Enabling Retrain-free Deep Neural Network Pruning Using Surrogate Lagrangian Relaxation

Deniz Gurevin[1*] , **Mikhail Bragin**[1*] , **Caiwen Ding**[2*] , **Shanglin Zhou**[2] , **Lynn Pepin**[2] ,
Bingbing Li[2] and **Fei Miao**[2]

[1] Department of Electrical and Computer Engineering, University of Connecticut, USA
[2] Department of Computer Science and Engineering, University of Connecticut, USA
{deniz.gurevin, mikhail.bragin, caiwen.ding, shanglin.zhou, lynn.pepin, bingbing.li,
fei.miao}@uconn.edu

Abstract

Network pruning is a widely used technique to reduce computation cost and model size for deep neural networks. However, the typical three-stage pipeline, i.e., training, pruning and retraining (fine-tuning) significantly increases the overall training trails. In this paper, we develop a systematic weight-pruning optimization approach based on Surrogate Lagrangian relaxation (SLR), which is tailored to overcome difficulties caused by the discrete nature of the weight-pruning problem while ensuring fast convergence.

We further accelerate the convergence of the SLR by using quadratic penalties. Model parameters obtained by SLR during the training phase are much closer to their optimal values as compared to those obtained by other state-of-the-art methods. We evaluate the proposed method on image classification tasks using CIFAR-10 and ImageNet, as well as object detection tasks using COCO 2014 and Ultra-Fast-Lane-Detection using TuSimple lane detection dataset. Experimental results demonstrate that our SLR-based weight-pruning optimization approach achieves higher compression rate than state-of-the-arts under the same accuracy requirement. It also achieves a high model accuracy even at the hard-pruning stage without retraining (reduces the traditional three-stage pruning to two-stage). Given a limited budget of retraining epochs, our approach quickly recovers the model accuracy.

1 Introduction

Deep neural network (DNN)-based statistical models are increasingly taxing of computational and storage resources, with costs proportional to the model size (i.e. the number of parameters in a model). This concern is especially pressing for embedded or IoT devices [Krizhevsky *et al.*, 2012; Simonyan and Zisserman, 2014]. By reducing model size, one reduces both storage costs and computation costs when evaluating a model. Various techniques exist for reducing

model size while maintaining performance of the model, e.g. weight pruning, sparsity regularization, quantization, and clustering. These techniques are collectively known as *model compression* [Dai *et al.*, 2017; Yang *et al.*, 2016; Molchanov *et al.*, 2017; Guo *et al.*, 2016; Tung *et al.*, 2017; Luo *et al.*, 2017; Ding *et al.*, 2017; Park *et al.*, 2017; Frankle and Carbin, 2018; He *et al.*, 2018; Liu *et al.*, 2019].

These works leverage the observation that training a compact model from scratch is more difficult and less performant than retraining a pruned model [Frankle and Carbin, 2018; Liu *et al.*, 2019]. Therefore, a typical three-stage pipeline, i.e., training (large model), pruning and retraining (also called "fine-tuning") is required. The pruning process is to set the redundant weights to zero and keep the important weights to best preserve the accuracy. The retraining process is necessary since the model accuracy dramatically drops after hard-pruning. However, this three-stage weight pruning technique significantly increases the overall training cost. For example, although the state-of-the-art weight pruning methods achieve very high compression rate while maintaining the prediction accuracy on many DNN architectures, the retraining process takes a longer time, e.g., 80 epochs for ResNet-18 on ImageNet, which is 70% of the original training epochs using Alternate Direction Method of Multipliers (ADMM) [Zhang *et al.*, 2018; Ren *et al.*, 2019].

Given the pros and cons of current weight pruning-method, this paper aims to answer the following questions: **Is there an optimization method that can achieve a high model accuracy even at the hard-pruning stage and can reduce retraining trails significantly? Given a limited budget of retraining epochs, is there an optimization method that can quickly recover the model accuracy (much faster than the state-of-the-art methods)?**

The major obstacle when answering these questions is the discrete nature of the model compression problems caused by "cardinality" constraints, which ensure that a certain proportion of weights is pruned.

In this paper, to address this difficulty, we develop a weight-pruning optimization approach based on recent Surrogate Lagrangian relaxation (SLR) [Bragin *et al.*, 2015], which overcomes all major convergence difficulties of standard Lagrangian relaxation. Within the SLR approach, Lagrangian multipliers approach their optimal values much faster as com-

---

*These authors contributed equally.

pared to those within other methods (e.g., ADMM).

We summarize our contributions/findings as:

- Tailoring the SLR-based approach to overcome difficulties caused by the discrete nature of the weight-pruning problem while ensuring fast convergence.

- Further acceleration of the SLR convergence by using quadratic penalties. The method possesses nice convergence properties inherited from fast accelerated reduction of constraint violation due to quadratic penalties and from the guaranteed convergence thereby leading to unparalleled performance as compared to other methods. Therefore, model parameters obtained by SLR are much closer to their optimal values as compared to those obtained by other state-of-the-art methods.

- Convergence proof of the SLR method for weight pruning problems. Existing coordination-based weight pruning methods do not converge when solving non-convex problems. Other coordination methods (e.g., ADMM) are not designed to handle discrete variables and other types of non-convexities.

- The achievement of high model accuracy even at the hard-pruning stage through our SLR-based weight-pruning optimization approach; given a limited budget of retraining epochs, the new method quickly recovers the model accuracy.

2 Related Works on Weight Pruning

Since lots of researchers have investigated that some portion of weights in neural networks are redundant, weight pruning is proposed to remove these less important coefficient values and it achieves model compression with similar performance compared to uncompressed one. Structured and non-structured (irregular) are two mainstream weight pruning methods.

In this work, we focus on irregular pruning which can achieve much higher accuracy than structured pruning [Wen et al., 2016] due to its flexibility in selecting weights.

Early work [Han et al., 2015] proposed an iterative and static magnitude-based weight pruning to explore the redundancy. Later, the compression rate has been further improved by integrating the ADMM [Boyd et al., 2011], an optimization algorithm that breaks optimization problems into smaller subproblems, where dynamic penalty is applied on all targeted weights each of which is then solved iteratively and more easily [Zhang et al., 2018; Niu et al., 2020]. [Louizos et al., 2018] proposed a framework for L_0 norm regularization for neural networks, which aim to prune the network during training by choosing weights setting them to exactly zero. Lottery ticket hypothesis was proposed by [Frankle and Carbin, 2018], which observes that a subnetwork of randomly-initialized network can replace the original network with the same performance.

3 Weight Pruning using SLR

Consider a DNN with N layers indexed by $n \in 1, ..., N$, where the weights at layer n are denoted by $\mathbf{W}_n$.

The objective is to minimize the loss function subject to constraints on the cardinality of weights within each layer n (the number of nonzero weights should be less than or equal to the predefined number l_n): $\min_{\mathbf{W}_n} f(\mathbf{W}_n) + \sum_{n=1}^{N} g_n(\mathbf{W}_n)$, where the first term represents the nonlinear smooth loss function and the other represents the non-differentiable "cardinality" penalty term [Zhang et al., 2018] with $g_n(\cdot)$ being the indicator function:

$$g_n(\mathbf{W}_n) = \begin{cases} 0 & \text{if } \text{card}(\mathbf{W}_n) \leq l_n, \ n = 1, \ldots, N, \\ +\infty & \text{otherwise.} \end{cases}$$

In its entirety, the problem cannot be solved either analytically or by using the stochastic gradient descent. To enable the decomposition into smaller manageable subproblems, duplicate variables are introduced and the problem is equivalently rewritten as: $\min_{\mathbf{W}_n} f(\mathbf{W}_n) + \sum_{n=1}^{N} g_n(\mathbf{Z}_n)$, subject to $\mathbf{W}_n = \mathbf{Z}_n, \ n = 1, \ldots, N$. To solve the problem, constraints are first relaxed by introducing Lagrangian multipliers and their violations are penalized by using quadratic penalties. The resulting *Augmented* Lagrangian function [Boyd et al., 2011; Zhang et al., 2018] of the above optimization problem is this given by

$$L_\rho(\mathbf{W}_n, \mathbf{Z}_n, \mathbf{\Lambda}_n) = f(\mathbf{W}_n) + \sum_{n=1}^{N} g_n(\mathbf{Z}_n)$$
$$+ \sum_{n=1}^{N} \text{tr}[\mathbf{\Lambda}_n^T(\mathbf{W}_n - \mathbf{Z}_n)] + \sum_{n=1}^{N} \frac{\rho}{2}\|\mathbf{W}_n - \mathbf{Z}_n\|_F^2, \quad (1)$$

where $\mathbf{\Lambda}_n$ is a matrix of Lagrangian multipliers (dual variables) corresponding to constraints $\mathbf{W}_n = \mathbf{Z}_n$, and has the same dimension as $\mathbf{W}_n$. The positive scalar ρ is the penalty coefficient, $\text{tr}(\cdot)$ denotes the trace, and $\|\cdot\|_F^2$ denotes the Frobenius norm.

In the following, motivated by decomposibility enabled by SLR [Bragin et al., 2015], which overcame all major difficulties of standard Lagrangian Relaxation, with much alleviated zigzagging and guaranteed convergence, the relaxed problem will be decomposed into two manageable subproblems, and the subproblems will then be efficiently coordinated by Lagrangian multipliers.

Step 1: Solve "Loss Function" Subproblem for $\mathbf{W}_n$ by using Stochastic Gradient Decent. At iteration k, for given values of multipliers $\mathbf{\Lambda}_n^k$, the first "loss function" subproblem is to minimize the Lagrangian function, while keeping $\mathbf{Z}_n$ at previously obtained values $\mathbf{Z}_n^{k-1}$ as

$$\min_{\mathbf{W}_n} L_\rho(\mathbf{W}_n, \mathbf{Z}_n^{k-1}, \mathbf{\Lambda}_n). \quad (2)$$

Since the regularizer is a differentiable quadratic norm, and the loss function is differentiable, the subproblem can be solved by stochastic gradient descent (SGD) [Bottou, 2010]. To ensure that multipliers updating directions are "proper," the following "surrogate" optimality condition needs to be satisfied following [Bragin et al., 2015, p. 179, eq. (12)]:

$$L_\rho(\mathbf{W}_n^k, \mathbf{Z}_n^{k-1}, \mathbf{\Lambda}_n^k) < L_\rho(\mathbf{W}_n^{k-1}, \mathbf{Z}_n^{k-1}, \mathbf{\Lambda}_n^k). \quad (3)$$

If (3) is satisfied, multipliers are updated following [Bragin *et al.*, 2015, p. 179, eq. (15)] as:

$$\mathbf{\Lambda}'^{k+1}_n := \mathbf{\Lambda}^k_n + s'^k(\mathbf{W}^k_n - \mathbf{Z}^{k-1}_n). \qquad (4)$$

where stepsizes are updated as [Bragin *et al.*, 2015, p. 180, eq. (20)] as

$$s'^k = \alpha^k \frac{s^{k-1}||\mathbf{W}^{k-1} - \mathbf{Z}^{k-1}||}{||\mathbf{W}^k - \mathbf{Z}^{k-1}||}. \qquad (5)$$

Step 2: Solve "Cardinality" Subproblem for $\mathbf{Z}_n$ through Pruning by using Projections onto Discrete Subspace.
The second "cardinality" subproblem is solved with respect to $\mathbf{Z}_n$ while fixing other variables at values $\mathbf{W}^k_n$ as

$$\min_{\mathbf{Z}_n} L_\rho\big(\mathbf{W}^k_n, \mathbf{Z}_n, \mathbf{\Lambda}'^{k+1}_n\big). \qquad (6)$$

Since $g_n(\cdot)$ is the indicator function, the globally optimal solution of this problem can be explicitly derived as [Boyd *et al.*, 2011]:

$$\mathbf{Z}^k_n = \mathbf{\Pi}_{\mathbf{S}_n}\big(\mathbf{W}^k_n + \frac{\mathbf{\Lambda}'^{k+1}_n}{\rho}\big), \qquad (7)$$

where $\mathbf{\Pi}_{\mathbf{S}_n}(\cdot)$ denotes the Euclidean projection onto the set $\mathbf{S}_n = \{\mathbf{W}_n \mid \mathrm{card}(\mathbf{W}_n) \le l_n\}, n = 1, \dots, N$.

To ensure that multipliers updating directions are "proper," the following "surrogate" optimality condition needs to be satisfied:

$$L_\rho\big(\mathbf{W}^k_n, \mathbf{Z}^k_n, \mathbf{\Lambda}'^{k+1}_n\big) < L_\rho\big(\mathbf{W}^k_n, \mathbf{Z}^{k-1}_n, \mathbf{\Lambda}'^{k+1}_n\big) \qquad (8)$$

Once (8) is satisfied,[1] multipliers are updated as:

$$\mathbf{\Lambda}^{k+1}_n := \mathbf{\Lambda}'^{k+1}_n + s^k(\mathbf{W}^k_n - \mathbf{Z}^k_n), \qquad (9)$$

where stepsizes are updated as

$$s^k = \alpha^k \frac{s'^k||\mathbf{W}^{k-1} - \mathbf{Z}^{k-1}||}{||\mathbf{W}^k - \mathbf{Z}^k||}, \qquad (10)$$

where stepsize-setting parameters [Bragin *et al.*, 2015, p. 188, eq. (67)] are:

$$\alpha^k = 1 - \frac{1}{M \times k^{(1-\frac{1}{k^r})}}, M > 1, 0 < r < 1. \qquad (11)$$

The algorithm of the new method is presented below:

---

[1]If condition (8) is not satisfied, the subproblems (2) and (6) are solved again by using the latest available values for $\mathbf{W}_n$ and $\mathbf{Z}_n$.

Algorithm 1 Surrogate Lagrangian Relaxation
1: Initialize $\mathbf{W}^0_n, \mathbf{Z}^0_n, \mathbf{\Lambda}^0_n$ and s^0
2: **while** Stopping criteria are not satisfied **do**
3:　　**1** solve subproblem (2),
4:　　**if** surrogate optimality condition (3) is satisfied **then**
5:　　　keep $\mathbf{W}^k_n, \mathbf{Z}^k_n$, and update multipliers $\mathbf{\Lambda}^k_n$ per (4) ,
6:　　**else**
7:　　　keep $\mathbf{W}^k_n, \mathbf{Z}^k_n$, do not update multipliers $\mathbf{\Lambda}^k_n$,
8:　　**end if**
9:　　**2** solve subproblem (6),
10:　**if** surrogate optimality condition (8) is satisfied **then**
11:　　keep $\mathbf{W}^k_n, \mathbf{Z}^k_n$, and update multipliers $\mathbf{\Lambda}^k_n$ per (9) ,
12:　**else**
13:　　keep $\mathbf{W}^k_n, \mathbf{Z}^k_n$, do not update multipliers $\mathbf{\Lambda}^k_n$,
14:　**end if**
15: **end while**

The theoretical results are summarized below:

Theorem. Sufficient Condition for Convergence of the Method: Assuming for any integer number κ there exists $k > \kappa$ such that surrogate optimality conditions (3) and (8) are satisfied, then under the stepsizing conditions (5) and (10)-(11), the Lagrangian multipliers converge to their optimal values $\mathbf{\Lambda}^*_n$ that maximize the following dual function:

$$q(\mathbf{\Lambda}) \equiv \min_{\mathbf{W}_n, \mathbf{Z}_n} L_\rho\big(\mathbf{W}_n, \mathbf{Z}_n, \mathbf{\Lambda}_n\big). \qquad (12)$$

Proof. The proof will be based on that of [Bragin *et al.*, 2015]. The major difference in the original SLR method [Bragin *et al.*, 2015] and the SLR method of this paper is the presence of quadratic terms within the Lagrangian function (1).

It should be noted that the weight pruning problem can be equivalently rewritten in a generic form as:

$$\min_{\mathbf{X}} \mathbf{F}(\mathbf{X}), s.t. \mathbf{G}(\mathbf{X}) = \mathbf{0}. \qquad (13)$$

where $\mathbf{X}$ collectively denotes the decision variables $\{\mathbf{W}_n, \mathbf{Z}_n\}$ and

$$\mathbf{F}(\mathbf{X}) \equiv f\big(\mathbf{W}_n\big) + \sum_{n=1}^N g_n(\mathbf{Z}_n) + \sum_{n=1}^N \frac{\rho}{2}||\mathbf{W}_n - \mathbf{Z}_n||^2_F,$$

$$\mathbf{G}(\mathbf{X}) \equiv \mathbf{W}_n - \mathbf{Z}_n, n = 1, \dots, N.$$

$$(14)$$

The feasible set of (13) is equivalent to that of the original model compression problem. Feasibility requires that $\mathbf{W}_n = \mathbf{Z}_n$, which makes the term $\frac{\rho}{2}||\mathbf{W}_n - \mathbf{Z}_n||^2_F$ within (13) disappear. Therefore, the Lagrangian function that corresponds to (13) is the *Augmented* Lagrangian function (1) to the original model compression problem. Furthermore, the surrogate optimality conditions (3) and (8) are the surrogate optimality conditions that correspond to the Lagrangian function $\mathbf{F}(\mathbf{X}) + \mathbf{\Lambda}\mathbf{G}(\mathbf{X})$ that corresponds to the problem (13). Therefore, since within the original SLR [Bragin *et al.*, 2015, Prop. 2.7, p. 188] convergence was proved under conditions on stepsizes (5) and (10)-(11) and the satisfaction of surrogate optimality conditions, both of which are satisfied here, multipliers converge to their optimal values for the model compression under consideration as well.　□

This is a major result in machine learning since this is the first application of the SLR method to guarantee theoretical convergence of the model compression problem while handling discrete variables as well as the quadratic terms. In fact, owing to quadratic terms, the method inherits nice convergence properties similar to those of Augmented Lagrangian Relaxation (ALR) (fast reduction of constraint violations) and to those of SLR (fast and guaranteed convergence without much of zigzagging of multipliers and the need of the so-called "optimal dual value" [Bragin *et al.*, 2015]) thereby leading to unparalleled performance as compared to other methods.

The SLR method enjoys the benefits of efficient subproblem solution coordination with guaranteed convergence made possible by stepsizes (5) and (10)-(11) approaching zero (without this requirement, multipliers (9) would not exhibit convergence), and by the satisfaction of surrogate surrogate optimality conditions (3) and (8) ensuring that multipliers are updated along "good" directions. In Section 4, it will be verified empirically that there always exists iteration κ after which the "surrogate" optimality conditions are satisfied thereby ensuring that the multipliers approach their optimal values during the entire iterative process.

The SLR method also benefits from the independent and systematic adjustment of two hyper-parameters: penalty coefficient ρ and the stepsize s^k. In contrast, other coordination methods are not designed to handle discrete variables and other types of non-convexities. For example, ADMM do not converge when solving non-convex problems [Boyd *et al.*, 2011, p. 73] because stepsizes ρ within the method does not converge to zero. Lowering stepsizes to zero within ADMM would also lead to decrease of the penalty coefficient, thereby leading to slower convergence.

4 Evaluation

In this section, we discuss our experimental results for image classification and object detection tasks.

4.1 Experimental Setup

All of the baseline models we use and our code in image classification tasks are implemented with PyTorch 1.6.0 and Python 3.6. For our experiments on COCO 2014 dataset, we used Torch v1.6.0, pycocotools v2.0 packages. For our experiments on TuSimple lane detection benchmark dataset[2], we used Python 3.7 with Torch v1.6.0, and SpConv v1.2 package. We conducted our experiments on Ubuntu 18.04 and using Nvidia Quadro RTX 6000 GPU with 24 GB GPU memory. We used 4 GPU nodes to train our models on the ImageNet dataset.

We start by pruning the pretrained models through SLR training. Afterwards, we perform hard-pruning on the model, completing the compression phase. We report the overall compression rate (or the percentage of remaining weights) and prediction accuracy.

[2]https://github.com/TuSimple/tusimple-benchmark

4.2 Evaluation on Image Classification Tasks

Models and Datasets. We used ResNet-18, ResNet-50, ResNet-56, ResNet-110 [He *et al.*, 2016] and VGG-16 [Simonyan and Zisserman, 2014] on CIFAR-10. On ImageNet ILSVRC 2012 benchmark, we use ResNet-18, ResNet-50 [He *et al.*, 2016] and MobileNetV2 [Sandler *et al.*, 2018]. We use the pretrained ResNet models on ImageNet from Torchvision's "models" subpackage. The accuracy of the pretrained baseline models we used are listed in Table 1.

Training Settings. In all experiments we used $\rho = 0.1$. In CIFAR-10 experiments, we used a learning rate of 0.01, batch size of 128 and ADAM optimizer during training. On ImageNet, we used a learning rate of 10^{-4}, batch size of 256 and SGD optimizer. For a fair comparison of SLR and ADMM methods, we used the same number of training epochs and sparsity configuration for both methods in the experiments.

Table 1 shows our comparison of SLR and ADMM on CIFAR-10 and ImageNet benchmark. Here, SLR parameters are set as $M = 300$, $r = 0.1$ and $s_0 = 10^{-2}$. After SLR and ADMM training, final hardpruning is performed and the hardpruning accuracy is reported without any additional retraining, given a limited budget of training epochs. According to our results, SLR outperforms ADMM method in terms of accuracy under the same compression rate. With higher compression rates, CIFAR-10 results show higher gap in accuracy between SLR and ADMM.

	Baseline (%)	Epoch	ADMM (%)	SLR (%)	Comp. Rate
CIFAR-10					
ResNet-18	93.33	40	72.84	*89.93*	8.71×
ResNet-50	93.86	50	78.63	*88.91*	6.57×
VGG-16	93.27	110	69.05	*87.31*	12×
ResNet-56	93.39	30	90.5	*92.3*	6.5×
ResNet-110	93.68	30	89.71	*92.31*	9.7×
ImageNet					
ResNet-18	69.7 / 89.0	40	58.9 / 81.7	*60.9 / 84.4*	6.5×
ResNet-50	76.1 / 92.8	30	64.8 / 85.1	*65.9 / 87.5*	3.89×
MobileNetV2	71.8 / 91.0	60	61.8 / 84.3	*63.2 / 85.5*	1.76×

Table 1: Comparison of SLR and ADMM on CIFAR-10 and ImageNet datasets. ImageNet results show Top-1 / Top-5 accuracy.

Figure 1 shows the hardpruning accuracy during SLR vs. ADMM on CIFAR-10 and ImageNet, corresponding to Table 1. During training, hardpruning accuracy is checked periodically. If the hardpruning accuracy meets the accuracy criteria, the training is stopped. As seen in the figures, SLR quickly converges and reaches the desired accuracy, almost $3\times$ faster than ADMM on CIFAR-10. Moreover, in Figure 1c, ADMM is still below the desired accuracy even after 300 epochs of training on VGG-16, while SLR completes training in 80 epochs. Similarly, as can be seen in Figures 1f and 1g, ADMM cannot output the desired accuracy after 60 and 50 epochs of training on ImageNet, while SLR reaches the threshold quickly.

Table 2 shows our comparison of SLR with other recent model compression works on CIFAR-10 benchmark. We report the percentage of parameters pruned after SLR training

(a) ResNet-18 on CIFAR-10.

(b) ResNet-50 on CIFAR-10.

(c) VGG-16 on CIFAR-10.

(d) ResNet-56 on CIFAR-10.

(e) ResNet-110 on CIFAR-10.

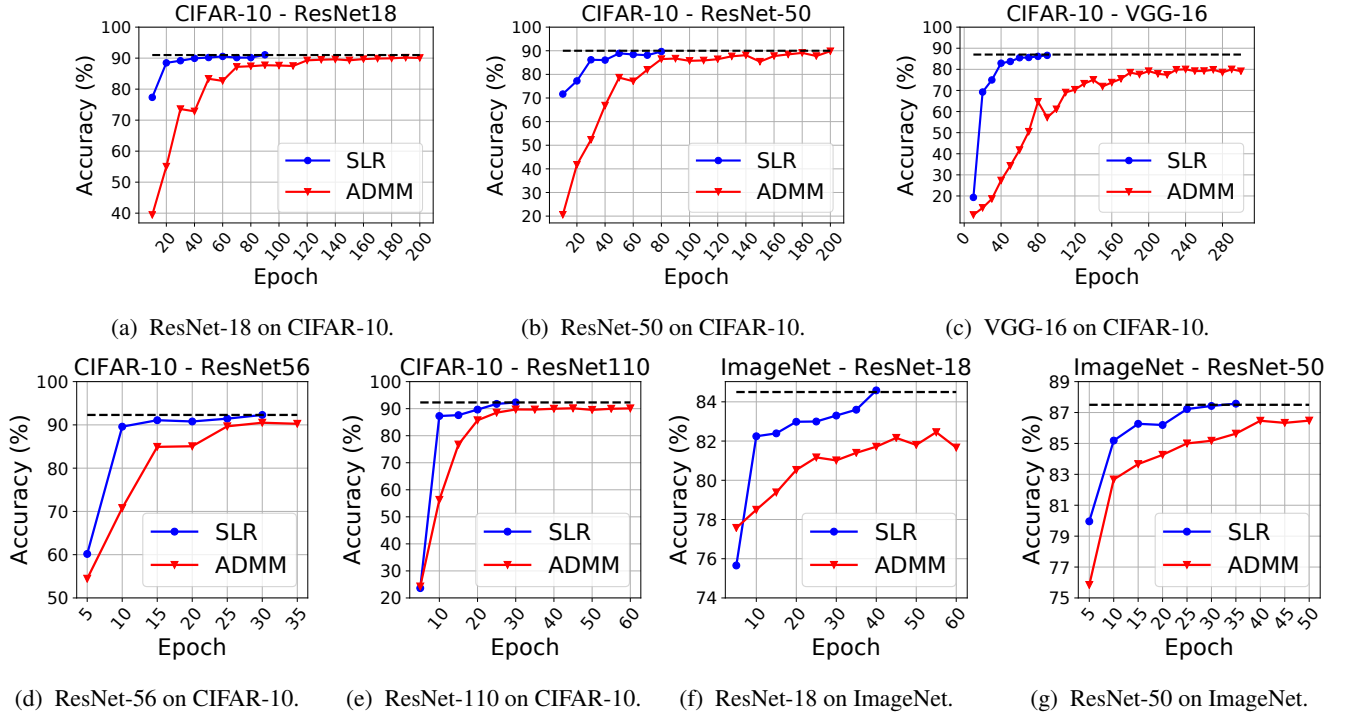(f) ResNet-18 on ImageNet.

(g) ResNet-50 on ImageNet.

Figure 1: Hardpruning accuracy after SLR and ADMM training on CIFAR-10 and ImageNet benchmarks. Accuracy is reported periodically and training is stopped when desired accuracy is reached.

Model	Method	Acc. (%)	Params Pruned (%)
VGG-16	**SLR**	91.2	
	AMC [He *et al.*, 2018]	91.0	90
	L0 [Louizos *et al.*, 2018]	80.0	
	SLR	93.1	60
	One-shot pruning[Liu *et al.*, 2019]	92.4	
	SLR	93.2	50
	Iter. Prun. [Han *et al.*, 2015]	92.2	
ResNet-18	SLR *(at 20k iterations)*	89.9	88.6
	Iter. prun. [Frankle and Carbin, 2018]	75.0	
ResNet-50	**SLR**	93.6	60
	AMC [He *et al.*, 2018]	93.5	
ResNet-56	**SLR**	92.3	84.4
	GSM [Ding *et al.*, 2019]	94.1	85.0
	Group Sparsity [Li *et al.*, 2020b]	92.65	79.2
	[Zhao *et al.*, 2019]	92.26	20.49
	GAL-0.6 [Lin *et al.*, 2019]	93.38	11.8
	[Li *et al.*, 2016]	93.06	13.7
	NISP [Yu *et al.*, 2018]	93.01	42.6
	KSE [Li *et al.*, 2019]	93.23	54.73
	DHP-50 [Li *et al.*, 2020a]	93.58	41.58

Table 2: SLR performance comparison with VGG-16, ResNet-18, ResNet-50 and ResNet-56 on CIFAR-10.

and the final accuracy. We start training the networks with a learning rate of 0.1 decreasing the learning rate by a factor of 10 at epochs 80 and 100. On ResNet-18, we compare our result at only 20k iterations. For VGG-16 and ResNet-50, we observe that SLR can achieve up to 60% pruning with less than 1% accuracy drop.

4.3 Evaluation on Object Detection Tasks

Models and Datasets. We used YOLOv3 and YOLOv3-tiny models [Redmon and Farhadi, 2018] on COCO 2014 benchmark. We used and followed the publicly available Ultralytics repository[3] for YOLOv3 and its pretrained models. For lane detection experiment, we used the pretrained model from Ultra-Fast-Lane-Detection [Qin *et al.*, 2020] on TuSimple lane detection benchmark dataset.

Training Settings. In all experiments we used $\rho = 0.1$. We set SLR parameters as $M = 300$, $r = 0.1$ and $s_0 = 10^{-2}$. We follow the same training settings provided by the repositories we use. Finally, we use the same number of training epochs and sparsity configuration for ADMM and SLR.

Testing Settings. On YOLOv3 models, we calculate the COCO mAP with IoU = 0.50 with image size of 640 for testing. On lane detection experiments, evaluation metric is "accuracy", which is calculated as $\frac{\sum_{\text{clip}} C_{\text{clip}}}{\sum_{\text{clip}} S_{\text{clip}}}$, where C_{clip} is the number of lane points predicted correctly and S_{clip} is the total number of ground truth in each clip.

Our comparison of SLR and ADMM methods on COCO dataset is shown in Table 3. We have compared the two methods under 3 different compression rates for YOLOv3-tiny and tested YOLOv3-SPP pretrained model with a compression rate of $1.98\times$. We can see that the model pruned with SLR method has higher accuracy after hard-pruning in all cases.

---

[3]https://github.com/ultralytics/yolov3

At a glance at YOLOv3-tiny results, we observe that the advantage of SLR is higher with an increased compression rate.

Architecture	Epoch	Method	Hardpruning mAP	Comp. Rate
YOLOv3-tiny (mAP = 37.1)	15	ADMM	35.2	1.19×
		SLR	*36.1*	
	20	ADMM	32.2	2×
		SLR	*36.0*	
	25	ADMM	25.3	3.33×
		SLR	*35.4*	
YOLOv3-SPP (mAP = 64.4)	15	ADMM	41.2	2×
		SLR	*53.2*	

Table 3: ADMM and SLR training results with YOLOv3 on COCO.

In compression rate $3.33\times$ on YOLOv3-tiny, given a limit of 25 epochs, we can observe that the gap between ADMM and SLR is much higher, which is the due to the faster convergence of SLR as shown in Figure 2b. Similarly, Figure 2a shows the mAP progress of YOLOv3 during SLR and ADMM training for 50 epochs, pruned with $2\times$ compression. SLR reaches the mAP threshold only at epoch 15.
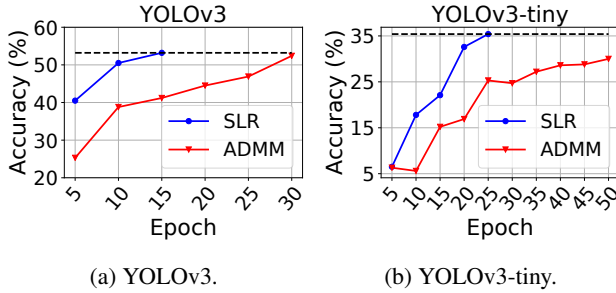

(a) YOLOv3. (b) YOLOv3-tiny.

Figure 2: Hardpruning accuracy of YOLOv3 and YOLOv3-tiny. Accuracy is reported every 5 epochs and training is stopped when methods reach the accuracy threshold.

Table 4 reports our result for the Lane Detection task on TuSimple lane detection benchmark after 40 epochs of training and 5 epochs of masked-retraining. We conducted experiments under 8 different compression rates. Similarly, Figure 3 illustrates the accuracy gap between ADMM and SLR methods after hard pruning as compression rate increases.

Compression	Hardpruning Acc. (%)		Retraining Acc. (%)	
	ADMM	SLR	ADMM	SLR
1.82×	**92.49**	*94.64*	94.28	94.63
2.54×	**92.25**	*94.56*	94.04	94.93
4.21×	**90.97**	*94.66*	94.18	94.68
12.10×	**88.41**	*94.51*	94.45	94.7
16.85×	**78.75**	*94.55*	94.23	94.65
22.80×	**67.79**	*94.62*	94.08	94.55
35.25×	**57.05**	*93.93*	93.63	94.34
77.67×	**46.09**	*89.72*	88.33	90.18

Table 4: SLR pruning results with ResNet-18 on TuSimple benchmark through different compression rates.
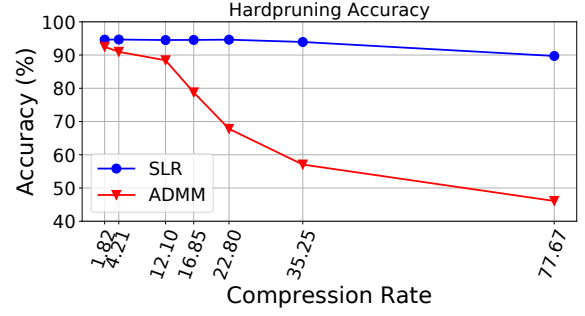

Figure 3: Hardpruning accuracy on TuSimple benchmark with ADMM vs. SLR training for several compression rates. SLR has a greater advantage over ADMM as compression rate increases.

From Figure 3, our observation is that for a small compression rate such as $1.82\times$, SLR has little advantage over ADMM in terms of hardpruning accuracy. However, as the compression rate increases, SLR starts to perform better than ADMM. For example, SLR survives $77.67\times$ compression with slight accuracy degration and results in 89.72% accuracy after hardpruning while ADMM accuracy drops to 46.09%. This demonstrates that our SLR-based training method has a greater advantage over ADMM especially in higher compression rates, as it achieves compression with less accuracy loss and reduces the time required to retrain after hard-pruning.

Finally, in Figure 4, we show the difference between weights of one layer before and after pruning with SLR and ADMM. In Figure 4a, we show the initial (non-pruned) weights and then show the sparsity of weights under the same compression rate ($77\times$) with SLR and ADMM. Initially, the layer has low sparsity. After training with SLR and ADMM, we can see an increased number of zeroed-weights. SLR moves towards the desired sparsity level faster than ADMM. In Figure 4b, we compare the sparsity of weights under the same accuracy (89.0%). It can be observed that SLR significantly reduced the number of non-zero weights and ADMM has more non-zero weights remaining compared with SLR.

4.4 Ablation Studies

We conducted several experiments to observe SLR behavior with respect to SLR parameters ρ, s_0, r and M on ResNet-18 model (93.33% accuracy) and CIFAR-10. We pruned the model through SLR training for 50 epochs with a compression rate of $8.71\times$ and observed the hardpruning accuracy every 10 epochs. Figure 5 shows the accuracy of the model through SLR training based on the different values of s_0, M and r. Based on the hardpruning accuracy throughout training, it can be seen that, even though the parameters do not have a great impact on the end result, choice of s_0 can impact the convergence of the model. From Figure 5a, we can state that $s_0 = 10^{-2}$ provides higher starting accuracy and converges quickly. Figure 5b and Figure 5c demonstrate the impact of M and r on the hardpruning accuracy respectively. Figure 6 demonstrates that there exists iteration κ (as required in the Theorem) so that the surrogate optimality condition, the high-level convergence criterion of the SLR method, is satisfied during training with $s_0 = 10^{-2}$, $\rho = 0.1$ thereby sig-

(a) Weights before after pruning with SLR (middle) and ADMM (right) under the same compression rate (77.6×).

(b) Weights before and after pruning with SLR (middle) and ADMM (right) under the same accuracy (89.0%).
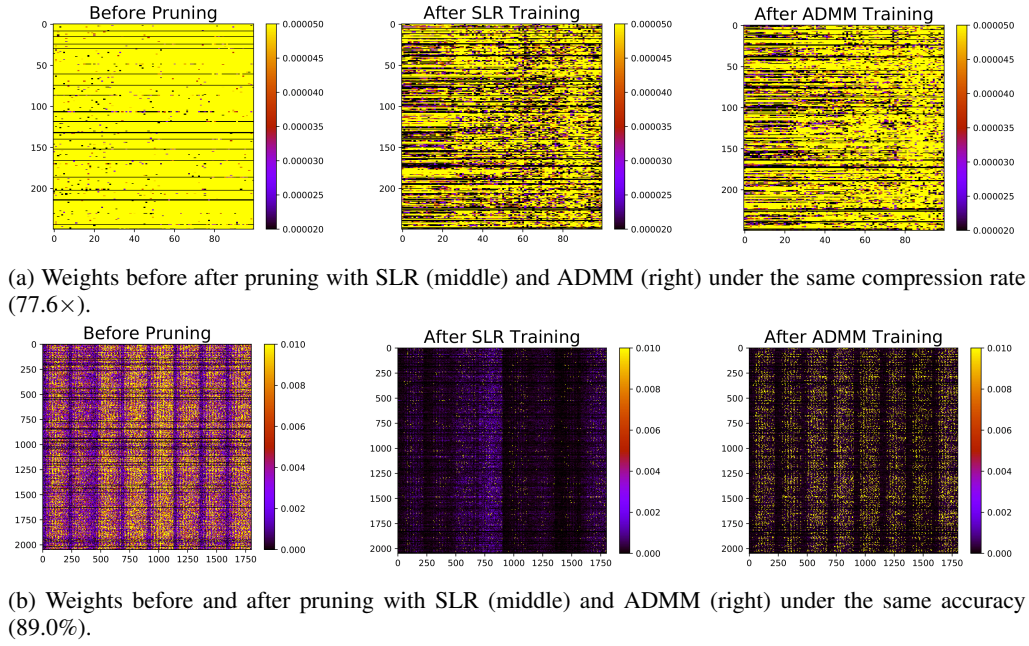
Figure 4: Heatmap of ResNet-18 weights on TuSimple benchmark before and after pruned with and ADMM. Weights are more zeroed out with SLR compared to ADMM.
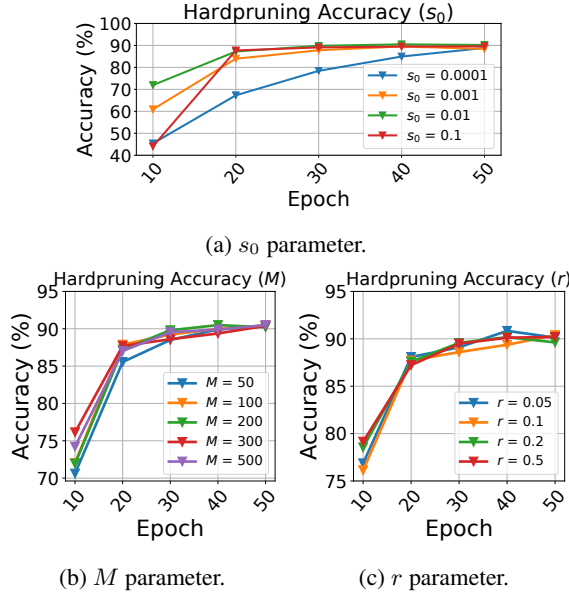

(a) s_0 parameter.

(b) M parameter.

(c) r parameter.

Figure 5: Hardpruning accuracy of ResNet-18 on CIFAR-10 during SLR training with respect to different values of s_0, M and r.

nifying that "good" multiplier-updating directions are always found. For example, after the conditions are violated at epoch 9, there exits $\kappa = 10$ so that at iteration 11, after $\kappa = 10$, the surrogate conditions are satisfied again.

5 Conclusions

In this paper, we presented the DNN weight-pruning problem as a non-convex optimization problem by adopting the
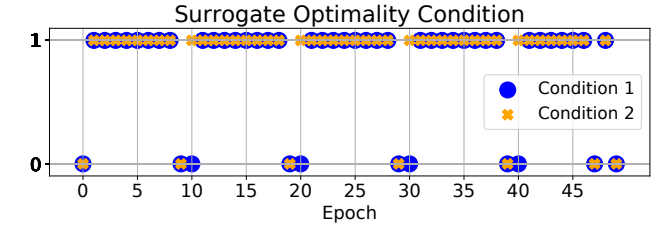

Figure 6: Surrogate optimality condition satisfaction graph during the SLR training of ResNet-18 on CIFAR-10 for 50 epochs (1: satisfied, 0: not satisfied). The condition is satisfied periodically.

cardinality function to induce weight sparsity. By using the SLR method, the relaxed weight-pruning problem is decomposed into subproblems, which are then efficiently coordinated by updating Lagrangian multipliers, resulting in fast convergence. We conducted weight-pruning experiments on image classification and object detection tasks to compare our SLR method against ADMM. We observed that SLR has a significant advantage over ADMM under high compression rates and achieves higher accuracy during weight pruning. SLR reduces the accuracy loss caused by the hard-pruning and so shortens the retraining process. With the effective optimization capabilities through coordination with clear advantages shown from several examples, the SLR method has a strong potential for more general DNN-training applications.

Acknowledgments

References

[Bottou, 2010] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *COMPSTAT*. Springer, 2010.

[Boyd *et al.*, 2011] S. P. Boyd, N. Parikh, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 3:1–122, 2011.

[Bragin *et al.*, 2015] M. A. Bragin, P. B. Luh, et al. Convergence of the surrogate lagrangian relaxation method. *Journal of Optimization Theory and Applications*, 164(1):173–201, 2015.

[Dai *et al.*, 2017] X. Dai, H. Yin, and N. K. Jha. Nest: A neural network synthesis tool based on a grow-and-prune paradigm. *arXiv preprint arXiv:1711.02017*, 2017.

[Ding *et al.*, 2017] C. Ding, S. Liao, Y. Wang, Z. Li, et al. Circnn: accelerating and compressing deep neural networks using block-circulant weight matrices. In *MICRO*, pages 395–408, 2017.

[Ding *et al.*, 2019] X. Ding, G. Ding, et al. Global sparse momentum sgd for pruning very deep neural networks. In *NeurIPS*, 2019.

[Frankle and Carbin, 2018] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2018.

[Guo *et al.*, 2016] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. In *NeurIPS*, pages 1379–1387, 2016.

[Han *et al.*, 2015] S. Han, J. Pool, et al. Learning both weights and connections for efficient neural network. In *NeurIPS*, pages 1135–1143, 2015.

[He *et al.*, 2016] K. He, X. Zhang, et al. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[He *et al.*, 2018] Y. He, J. Lin, et al. Amc: Automl for model compression and acceleration on mobile devices. In *ECCV*, September 2018.

[Krizhevsky *et al.*, 2012] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, pages 1097–1105, 2012.

[Li *et al.*, 2016] H. Li, A. Kadav, et al. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[Li *et al.*, 2019] Y. Li, S. Lin, et al. Exploiting kernel sparsity and entropy for interpretable cnn compression. *CVPR*, 2019.

[Li *et al.*, 2020a] Y. Li, S. Gu, et al. Dhp: Differentiable meta pruning via hypernetworks. *ArXiv*, abs/2003.13683, 2020.

[Li *et al.*, 2020b] Y. Li, S. Gu, et al. Group sparsity: The hinge between filter pruning and decomposition for network compression. *CVPR*, 2020.

[Lin *et al.*, 2019] S. Lin, R. Ji, et al. Towards optimal structured cnn pruning via generative adversarial learning. *CVPR*, 2019.

[Liu *et al.*, 2019] Z. Liu, M. Sun, et al. Rethinking the value of network pruning. In *ICLR*, 2019.

[Louizos *et al.*, 2018] C. Louizos, M. Welling, and D. P. Kingma. Learning sparse neural networks through l0 regularization. In *ICLR*, 2018.

[Luo *et al.*, 2017] J. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, pages 5068–5076. IEEE, 2017.

[Molchanov *et al.*, 2017] D. Molchanov, A. Ashukha, and D. Vetrov. Variational dropout sparsifies deep neural networks. In *ICML*, pages 2498–2507, 2017.

[Niu *et al.*, 2020] W. Niu, X. Ma, et al. Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. In *Proc. of ASPLOS*, pages 907–922, 2020.

[Park *et al.*, 2017] E. Park, J. Ahn, and S. Yoo. Weighted-entropy-based quantization for deep neural networks. In *CVPR*, 2017.

[Qin *et al.*, 2020] Z. Qin, H. Wang, and X. Li. Ultra fast structure-aware deep lane detection. *arXiv preprint arXiv:2004.11757*, 2020.

[Redmon and Farhadi, 2018] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[Ren *et al.*, 2019] Ao Ren, Tianyun Zhang, et al. Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers. In *ASPLOS*, pages 925–938, 2019.

[Sandler *et al.*, 2018] M. Sandler, A. G. Howard, et al. Mobilenetv2: Inverted residuals and linear bottlenecks. *CVPR*, 2018.

[Simonyan and Zisserman, 2014] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[Tung *et al.*, 2017] F. Tung, S. Muralidharan, and G. Mori. Fine-pruning: Joint fine-tuning and compression of a convolutional network with bayesian optimization. *arXiv preprint arXiv:1707.09102*, 2017.

[Wen *et al.*, 2016] W. Wen, C. Wu, et al. Learning structured sparsity in deep neural networks. In *NeurIPS*, pages 2074–2082, 2016.

[Yang *et al.*, 2016] T. Yang, Y. Chen, and V. Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. *arXiv preprint arXiv:1611.05128*, 2016.

[Yu *et al.*, 2018] R. Yu, A. Li, et al. Nisp: Pruning networks using neuron importance score propagation. *CVPR*, 2018.

[Zhang *et al.*, 2018] T. Zhang, S. Ye, et al. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *ECCV*, pages 184–199, 2018.

[Zhao *et al.*, 2019] C. Zhao, B. Ni, et al. Variational convolutional neural network pruning. *2019 CVPR*, 2019.